Gemini
Controls
Group
Interface
Controls
Document

ICD 18 - The Gemini World Coordinate System

Steven Beard
20 November 1998

This document provides a brief overview of the World Coordinate System used by Gemini.

1.0 Introduction

1.1 Acronyms & Abbreviations

EPICS	Experimental Physics and Industrial Control System
ICS	Instrument Control System
IOC	Input Output Controller
TCS	Telescope Control System
WCS	World Coordinate System

1.2 References

(1) TN-PS-G0045, "Astronomical Coordinate Frames I: Astrometry" (tcs_ptw_008)
(2) TN-C-G0070, "The Flow of WCS Information through a Gemini ICS" (cics_smb_042)
(3) TN-PS-G0071, "Astronomical Coordinate Frames II: Quick Look Support" (tcs_ptw_009)

2.0 World Coordinate Systems for Dummies

The following text is taken from an email message and some example code which was circulated to interested parties on 20 November 1998. The reader should consult document (1) for a more thorough description of the use of World Coordinates.

Pat Wallace has explained to me the theory behind art of World Coordinate Systems, and I think I now have a basic understanding of how an imaging instrument can generate the World Coordinate System header information it needs to provide. The reference documents give all the details. I found these documents hard to digest at first because they contain so much information, but Pat has pointed me at the salient points most useful to a Gemini instrument controller developer. The most useful part of these documents as far as an instrument programmer is concerned is the sky imager example in TN-PS-G0045, section 5.2. This example provides everything an imaging Gemini instrument needs. I was confused by this example when I first saw it because it contains code designed to run under Unix, so much of the code shown is devoted to simulating the TCS time system and WCS context.

I have adapted this example, with Pat's help, to show how WCS information can be obtained on an EPICS IOC. You will find this example later in this document. The example consists of two functions:

wcsCalibrate - Simulates a calibration measurement by loading some arrays full of numbers, then uses those numbers to obtain an (i,j) to (x,y) fit.

In practice someone would actually calibrate an instrument and store these lists of numbers into a file. (See the comments in the code for details).

It is this calibration that makes the example valid for any imager, regardless of how many reflections, rotations, magnifications or distortions it makes.

wcsTest - Gets the current TCS context and the (i,j) to (x,y) fit and calculates the WCS header items.

The example assumes that you have a genSub record executing regularly in the background that reads the context information from the TCS. Section 3.2 of TN-PS-G0045 explains how to do this. I recommend one change to this example. I would enhance the genSub code to look like this:

```
#include "alarm.h"

long updateAstCtx(struct gensubRecord *pgsub)
{
   if ( pgsub->sevr != ALARM_INVALID )
   {
      (void) astSetctx ( pgsub->a );
   }
   else
   {
      printErr ("ERROR - TCS disconnected");
   }
}
```

That way you get an error message if the TCS database is not available. In practice you may want to replace the "printErr" with your own error reporting method. I also arranged to have the message appear only once when the first disconnection is noticed.

It should be possible to extend the WCS calculation to spectrographs, but the details of how to do this have not yet been decided. Pat tells me that Figure 1 in TN-PS-G0071 encapsulates the situation for both imagers and spectrographs.

3.0 Example Code

```
/*
 * This file contains some code for testing the getting and setting of
 * World Coordinate System (WCS) information, as described in document
 * tcs_ptw_008, "World Coordinates, Part I: Astrometry". The code has
 * been adapted from the example given in section 5.2 of that paper.
 *
 * Simulated calls to obtain TCS context information have been replaced by
 * real TCS function calls.
 *
 * The code assumes that a TCS World Coordinate System context is available
 * and has been set elsewhere using astSetCtx(). In practise this is done
 * in a genSub record which grabs the WCS context from the TCS at regular
 * intervals. The TCS database needs to be up and running and accessible.
 *
 * Steven Beard, 18 November 1998.
 */

#include <vxWorks.h>
#include <stdio.h>
#include <stdlib.h>
#include <sysLib.h>
#include <taskLib.h>

#include "timeLib.h"
#include "slalib.h"
#include "astLib.h"

/* Define global constants */

#define  NPOINTS           4    /* Number of measured points             */
                                /* (must be at least 3).                 */
#define  MATRIXSIZE        6    /* Size of transformation matrix         */
                                /* (always 6).                           */

/* Define global variables */

double   fpxy[NPOINTS][2];     /* Array of defined focal plane XY       */
                               /* coordinates.                          */
double   pixij[NPOINTS][2];    /* Array of measured pixel IJ            */
                               /* coordinates (unbinned and unwindowed). */
double   detij[NPOINTS][2];    /* Array of binned and windowed pixel IJ  */
                               /* coordinates corresponding to the same  */
                               /* points on the detector.               */
double   cij[MATRIXSIZE];      /* XY to IJ transformation matrix.       */

/*
 * Function wcsCalibrate() defines a pixel coordinate (i,j) to (x,y)
 * transformation.
 *
 * In practise this will be done by executing the following steps:
 *
 * Step 1 - calibration
 *
 *    For each calibration point, p
```

```
 *        Command the TCS to put a star at position (x,y) in the focal plane,
 *           (fpxy[p][0]), fpxy[p][1]).
 *        Measure the position of that star on the detector in pixels (i,j),
 *           (pixij[p][0], pixij[p][1]).
 *    Next point (at least 3 points are needed).
 *
 *    In this example the points are defined arbitrarily. In the real world
 *    the (x,y) and (i,j) measurements will be stored in a file and read when
 *    needed. 4 points are used merely as an example and a more reiable fit
 *    would be obtained using more points. A fit such as this is valid for any
 *    imaging instrument, not just the HRWFS.
 *
 * Step 2 - transform the (i,j) measurements to match the current
configuration
 *
 *    It is assumed the calibration at step (1) will have been done with the
 *    detector generating unbinned full frames. If the detector is
reconfigured
 *    to bin the data or generate a smaller region of interest, the (i,j)
 *    values need to be transformed to match the detector coordinates in the
 *    new frame of reference.
 *
 *    For example, if the detector is skipping the first "iskip" pixels and
 *    binning  the remaining pixels by a factor "ibin" each i measurement
 *    becomes
 *
 *        i_new = ((i_old - 0.5 - iskip) / ibin) + 0.5;
 *
 * Step 2 - fit
 *
 *    Function astFitij() is used to generate an (i,j) to (x,y)
transformation.
 *    This transformation will be valid as long as the parameters used for
 *    steps (1) and (2) remain the same.
 *
 *    If the detector is reprogrammed (e.g. different binning or window)
 *    go back to step (2).
 *    If the instrument is reconfigured (e.g. different camera used)
 *    go back to step (1).
 *
 */

STATUS wcsCalibrate (void)
{
    int    wcsStat;         /* WCS status.                                  */
    int    p;               /* Point counter.                               */

    int    ibin, jbin;      /* Detector i and j binning factors.            */
    int    iskip, jskip;    /* Detector will skip this number of pixels     */
                            /* in i and j.                                  */

    double pixis;           /* x to i scale factor.                         */
    double pixjs;           /* y to j scale factor.                         */
    double perp;            /* Non-perpendicularity of i and j axes in      */
                            /* radians.                                     */
    double orient;          /* Orientation of (i,j) axes with respect to    */
                            /*  (x,y) in radians.                           */
```

```
/*
 * Step 1.
 *
 * Define an example of a set of calibration measurements, for
 * demonstration purposes.
 *
 * NOTE: In reality the points would be read from a file and counted,
 * with an end of file signifying the end of the list.
 */

pixij[0][0] = 51.3;
pixij[0][1] = 49.5;
fpxy[0][0] = 19.15;
fpxy[0][1] = 21.31;

pixij[1][0] = 50.7;
pixij[1][1] = 1227.8;
fpxy[1][0] = -23.77;
fpxy[1][1] = -13.95;

pixij[2][0] = 2179.6;
pixij[2][1] = 1230.4;
fpxy[2][0] = 23.36;
fpxy[2][1] = -22.26;

pixij[3][0] = 2182.3;
pixij[3][1] = 53.1;
fpxy[3][0] = 28.08;
fpxy[3][1] = 3.90;

/*
 * Step 2.
 *
 * Transform the measurements using the current detector controller
 * configuration.
 */

ibin  = 1;      /* Default values. */
jbin  = 1;
iskip = 0;
jskip = 0;

printf ("Enter detector binning factors in i and j: ");
scanf ("%d %d", &ibin, &jbin);

if ( ibin <= 0 ) ibin = 1;
if ( jbin <= 0 ) jbin = 1;

printf ("Enter number of pixels detector skips in i and j: ");
scanf ("%d %d", &iskip, &jskip);

for (p=0; p<NPOINTS; p++)
{
    detij[p][0] = ((pixij[p][0] - 0.5 - (double) iskip) /
                  (double) ibin) + 0.5;
```

```
        detij[p][1] = ((pixij[p][1] - 0.5 - (double) jskip) /
                       (double) jbin) + 0.5;
    }

    /*
     * Display the calibration points entered.
     */

    printf ("Calibration points:\n");
    printf ("i      \tfpx      \tfpy      \tpixi      \tpixj      "
            "\tdeti      \tdetj\n");

    for (p=0; p<NPOINTS; p++)
    {
        printf ("%d \t%f \t%f \t%f \t%f \t%f \t%f\n", p,
                fpxy[p][0], fpxy[p][1],
                pixij[p][0], pixij[p][1],
                detij[p][0], detij[p][1]);
    }

    /*
     * Define the (i,j) to (X,Y) transformation. The transformation is written
     * to matrix cij.
     */

    wcsStat = astFitij (NPOINTS, fpxy, detij, cij, &pixis, &pixjs,
                        &perp, &orient);
    if (wcsStat != 0)
    {
        printErr ("astFitij: Failed to define (i,j) to (X,Y) transformation."
                  " Status=%d\n", wcsStat);
        return (ERROR);
    }

    printf ("Best fit scale is %f X units per i pixel and "
            "%f Y units per j pixel\n", pixis, pixjs);
    printf ("i/j non-perpendicularity is %f radians.\n", perp);
    printf ("i/j is rotated by %f radians with respect to x/y axis.\n",
            orient);

    printf ("Cij matrix contains %f %f %f %f %f %f\n", cij[0], cij[1], cij[2],
            cij[3], cij[4], cij[5]);

    return (OK);
}

/*
 * Function wcsTest() generates FITS header values describing the current
World
 * Coordinate System. It assumes that function wcsCalibrate() has been
executed
 * and a (i,j) to (X,Y) transformation matrix has been defined in global
 * variable cij.
 */

STATUS wcsTest (void)
{
```

```
    int             wcsStat;          /* WCS status.                         */
    struct WCS_CTX ctx;               /* World Coordinate System context.    */
    struct WCS     wcs;               /* Basic TCS World Coordinate System.  */
    struct WCS     wcsij;             /* World Coordinate System transformed */
                                      /* into detector frame of reference.   */

    FRAMETYPE       trackFrame;       /* TCS track frame.                    */
    struct EPOCH   trackEquinox;      /* TCS track equinox.                  */

    double          rawTimeStamp;     /* Time stamp in Gemini raw time.      */
    double          timeTAI;          /* International atomic time in secs.  */
    double          trackWavelength;  /* Track wavelength in microns.        */
    double          rawTimeWcs;       /* Gemini raw time at which WCS info   */
                                      /* valid.                              */

    int             chopState;        /* Chop state.                         */

    char            ctype1[81];       /* World Coordinate System projection  */
                                      /* type for axis 1.                    */
    double          crpix1;           /* Pixel coordinate reference for      */
                                      /* axis 1.                             */
    double          crval1;           /* World coordinate reference for      */
                                      /* axis 1.                             */
    char            ctype2[81];       /* World Coordinate System projection  */
                                      /* type for axis 2.                    */
    double          crpix2;           /* Pixel coordinate reference for      */
                                      /* axis 2.                             */
    double          crval2;           /* World coordinate reference for      */
                                      /* axis 2.                             */
    double          cd1_1;            /* xi rotation/skew matrix element.    */
    double          cd1_2;            /* xj rotation/skew matrix element.    */
    double          cd2_1;            /* yi rotation/skew matrix element.    */
    double          cd2_2;            /* yj rotation/skew matrix element.    */
    char            radecsys[81];     /* Type of RA/Dec (for celestial       */
                                      /* coordinates).                       */
    double          equinox;          /* Epoch of mean equator & equinox     */
                                      /* (celestial  coords).                */
    double          mjdobs;           /* Modified Julian Date.               */

    /*
     * Get a timestamp for the World Coordinate System information.
     * This timestamp should be obtained close to the start of the observation
     */

    if (timeNow (&rawTimeStamp) != OK)
    {
        printErr ("Failed to get time stamp.\n");
        return (ERROR);
    }

    printf ("Gemini raw time = %f\n", rawTimeStamp);

    /*
     * Convert the start time into International Atomic Time (TAI).
     * This time will be used to generate the MJD-OBS field in the FITS
header.
     *
```

```
 * There is currently no internationally agreed standard defining the
 * timescale for MJD-OBS. TAI is used here because it is a sensible choice
 * and, in fact, was once specified in a draft standard in July 1996 that
 * was subsequently withdrawn.
 * Whatever timescale is specified here, it is important that it be
 * continuous across a leap second. Suitable alternatives are
 * Terrestrial Time (TT) and Universal Time 1 (UT1). UTC is NOT suitable.
 *
 * NOTE: Pat Wallace is checking this with the FITS committee.
 */

if (timeThenD (rawTimeStamp, TAI, &timeTAI) != OK)
{
    printErr ("Failed to convert time stamp to TAI.\n");
    return (ERROR);
}

printf ("TAI = %f\n", timeTAI);


/*
 * Obtain the current TCS context from the locally stored copy.
 * This assumes that a TCS context has been obtained elsewhere and stored
 * using astSetCtx(), as described in section 3 of document tcs_ptw_008.
 */

wcsStat = astGetctx (&ctx);
if (wcsStat != 0)
{
    printErr ("astGetctx: Failed to get current WCS context. Status=%d\n",
               wcsStat);
    return (ERROR);
}

/*
 * Set the required tracking frame, equinox and wavelength in microns to
 * sensible defaults. These parameters define the coordinate frame in
 * which you would like the WCS information to appear, and in practise it
 * makes sense to make them the same as the tracking frame, equinox and
 * wavelength the TCS is using. These numbers can be obtained by querying
 * the TCS Status Alarm Database from the EPICS SIR records described in
 * ICD 3.1/1.1.11.
 */

trackFrame        = FK5;
trackEquinox.type = 'J';  /* NOTE: The data type is 'char'. */
trackEquinox.year = 2000.0;
trackWavelength   = 0.5;

/*
 * Set the chop state to which the WCS coordinate information refers.
 */

chopState = 0;              /* 0 means chop state A. */

/*
 * Extract the current focal plane to sky WCS transformation from the TCS
```

```
     * context.
     *
     * NOTE: If an instrument needs more than one focal plane to sky
     * transformation (for example for different wavelengths or chop states)
     * this function will need to be called more than once.
     */

    wcsStat = astCtx2tr (ctx, trackFrame, trackEquinox, trackWavelength,
                         chopState, &wcs, &rawTimeWcs);
    if (wcsStat != 0)
    {
        printErr ("Failed to get focal plane to sky WCS transformation. "
                  "Status = %d\n", wcsStat);
        return (ERROR);
    }

    printf ("WCS information extracted from TCS context is valid at time
%f\n",
            rawTimeWcs);

    /*
     * Combine the (i,j) to (x,y) model, cij, and (x,y) to (RA,Dec) model,
wcs,
     * into a single (i,j) to (RA,Dec) model, wcsij.
     *
     * NOTE: If an instrument needs to apply further coordinate
transformations
     * they can be done here. (The exception is the simple binning of pixels
on
     * the detector, which is more easily done by changing the (i,j)
     * calibration measurements.
     */

    wcsStat = astXtndtr ( cij, wcs, &wcsij );
    if (wcsStat != 0)
    {
        printErr ("astXtndtr: Failed to combine i-j to x-y and x-y to "
                  "RA-Dec models. Status = %d\n", wcsStat);
        return (ERROR);
    }

    /*
     * Calculate the FITS header values.
     *
     * NOTE: The function astFITS can generate FITS header strings directly.
     * I am using astFITSv because the Gemini DHS requires WCS information to
     * be associated with data by means of attribute/value pairs. The name and
     * value of each attribute is defined using the dhsBdAttribAdd() function,
     * as described in ICD 3.
     */

    wcsStat = astFITSv (wcsij, trackFrame, trackEquinox, timeTAI,
                        ctype1, &crpix1, &crval1,
                        ctype2, &crpix2, &crval2,
                        &cd1_1, &cd1_2, &cd2_1, &cd2_2,
                        radecsys, &equinox, &mjdobs);
    if (wcsStat != 0)
```

```
    {
        printErr ("Failed to calculate FITS header. Status = %d\n", wcsStat);
        return (ERROR);
    }

    printf ("World Coordinate System Header\n");
    printf ("------------------------------\n");
    printf ("ctype1   = %s\n", ctype1);
    printf ("crpix1   = %f pixels\n", crpix1);
    printf ("crval1   = %f degrees = %f hours\n", crval1,
            (crval1 / (double) 15.0));
    printf ("ctype2   = %s\n", ctype2);
    printf ("crpix2   = %f pixels\n", crpix2);
    printf ("crval2   = %f degrees\n", crval2);
    printf ("cd1_1    = %f\n", cd1_1);
    printf ("cd1_2    = %f\n", cd1_2);
    printf ("cd2_1    = %f\n", cd2_1);
    printf ("cd2_2    = %f\n", cd2_2);
    printf ("radecsys = %s\n", radecsys);
    printf ("equinox  = %f\n", equinox);
    printf ("mjd-obs  = %f\n", mjdobs);

    return (OK);
}
```